A New CRT-RSA Algorithm Resistant to Powerful Fault Attacks

Nevine Ebeid
Certicom Corp.
a Subsidiary of Research In Motion Limited
Mississauga, Ontario, Canada
nebeid@certicom.com

Rob Lambert
Certicom Corp.
a Subsidiary of Research In Motion Limited
Mississauga, Ontario, Canada
rlambert@certicom.com

ABSTRACT

CRT-RSA is widely deployed in embedded devices to accelerate the RSA signature generation by about four times compared to regular RSA. However, since the Bellcore attack of 1996, research into securing CRT-RSA has remained active as countermeasures are themselves attacked. In this paper, we propose a new countermeasure designed with a powerful attacker in mind. The attacker may inject multiple precise/random faults and may alter the program counter to skip one or more instructions.

The strength of our countermeasure derives from combining signature validation with signature unblinding modulo n.

Keywords: RSA, Chinese Remainder Theorem, Fault injection attacks, Countermeasures.

1. INTRODUCTION

Securing the CRT-RSA signature scheme against fault attacks (FA) has attracted the attention of several researchers since the Bellcore attack was introduced in 1996 by Boneh *et al.* [8]. In general, fault attacks aim to produce an erroneous result in one or more steps of a cryptographic computation. Collecting partially erroneous results or faulty and correct results from the same computation can help the attacker deduce secret information and possibly break the system.

The immediate objective of inducing a fault may be to change some bit values in memory locations or internal registers, and may also involve changing the values of the program counter or stack pointer and, by this means, skipping some or all of the remaining steps in the algorithm. The fault may also attempt to toggle the result of a conditional check [20]. There are various ways to induce faults in a device while performing a cryptographic application. Arbitrary faults may be injected by varying temperature, the external clock, the supply voltage, or more accurately targeted faults may be injected with laser, X-ray or ion beams [4].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WESS'10, October 24, 2010, Scottsdale, AZ,USA
Copyright 2010 ACM 978-1-4503-0078-0 ...$10.00.

The CRT-RSA signature algorithm has become a well-known target of these attacks. Consisting of two lengthy exponentiations modulo each of the secret factors of the RSA modulus, it is relatively easy to induce a fault in just one of these two operations of CRT-RSA. Collecting a single faulty signature, an attacker may factor the modulus, sidestepping this computationally hard problem on which the mathematical security of the RSA scheme is based.

Since the publication of this attack, several countermeasures have been proposed and many attacks on these countermeasures have also been developed. Countermeasures for CRT-RSA often avoid the use of e, which is a public component of the RSA public key, since it is not one of the personalization parameters p, q, d_p, d_q and $q^{-1} \bmod p$. However, some of theses countermeasures require d, the private exponent, as part of a precomputation phase, which is also not a personalization parameter [5, 6, 21]. Conventionally, e is defined to be a relatively small integer, allowing the signature verification to be more efficient than signature generation. In this paper, we investigate what is possible if e is available. We present a new approach which, employing e, blinds the whole CRT-RSA computation, and which includes an implicit blinded signature validation modulo n in the unblinding computation.

In Sect. 2, we provide a brief overview of attacks and the previously proposed countermeasures, we also discuss their vulnerability to fault attacks under our attack model. In Sect. 3, we present our new CRT-RSA computation, interwoven with countermeasures against fault analysis attacks. The security and performance evaluation of the proposed algorithm is presented in Sect. 4 and Sect. 5. Finally, we provide our conclusions in Sect. 6.

2. FAULT ATTACKS ON CRT-RSA AND PROPOSED COUNTERMEASURES

In this section, we review the Bellcore attack on CRT-RSA and the various countermeasures that have been proposed.

2.1 CRT-RSA signature scheme

In the RSA signature scheme [25], party A owns a long-term *public key* (n, e) and a corresponding *private key* (p, q, d), where $n = pq$; p and q are large primes, e is the *public exponent*; $1 < e < \phi(n) = (p-1)(q-1)$, $\gcd(e, \phi(n)) = 1$ and $d = e^{-1} \bmod \phi(n)$ is the *private exponent.*

To sign a message m, or rather a hashed and/or padded version thereof, A calculates the signature $s = m^d \bmod n$ and sends it to B who, using A's public key, can confirm A's signature on m by verifying that $m = s^e \bmod n$.

An approximately four times more efficient computation was suggested by Quisquater and Couvreur [24] using the *Chinese Remainder Theorem* (CRT), where A performs the following steps to compute s

$$
\begin{aligned}
s_p &= m^{d_p} \bmod p \ , && (1)\\
s_q &= m^{d_q} \bmod q \ , && (2)\\
s &= \mathrm{CRT}(s_p, s_q) \ , &&\\
&= s_q + q \cdot (i_q \cdot (s_p - s_q) \bmod p) \ , && (3)
\end{aligned}
$$

where $i_q = q^{-1} \bmod p$. Here (3) is referred to as Garner's CRT recombination [15].

2.2 The Bellcore Attack

Bellcore researchers [8] have presented a fault attack on CRT-RSA as follows: A fault is injected in the device during the computation of (1) (equivalently (2)), resulting in a faulty value $\underline{s_p}$ ($\underline{s_q}$), and hence a faulty signature $\underline{s}$. Now we have

$$
\begin{aligned}
\underline{s_p}^{\,e} &\not\equiv m \pmod p \ ,\\
s_q^e &\equiv m \pmod q \ , \qquad (4)\\
\gcd(\underline{s}^{\,e} - m, n) &= q \ ,
\end{aligned}
$$

thus factoring n and solving the computationally hard problem underlying RSA. This attack is also referred to as the *gcd attack* [21].

2.3 Previous Countermeasures

Several countermeasures to the gcd attack have been proposed (we refer the reader to [20] for a chronological overview). The earlier countermeasures were based on inserting one or more conditional checks in the algorithm to verify the integrity of the values [2, 10, 17, 18, 27].

Yen *et al.* [32] noted that fault injection on status register flags can bypass conditional checks in countermeasures. They hence introduced the concept of *infective computation* which aims at infecting the resulting signature, *i.e.*, rendering it unusable for the attacker in the case where a fault is injected in one of the two exponentiations. Several subsequent countermeasures [5, 6, 11] employed infective computation.

Kim and Quisquater [20] injected a second fault with the intent of skipping the final step of the algorithm where the signature is validated and were thereby able to retrieve the faulty signature and perform a Bellcore attack. Therefore, they proposed keeping the signature *blinded* until it is validated and perform the validation in a way such that it infects the signature if it was faulty.

Another technique was proposed by Boscher, Naciri and Prouff [10] and later slightly modified by Boscher, Handschuh and Trichina [9]. It is based on the checks performed after the right-to-left exponentiation is executed and a similar check after the CRT recombination of the signature components. The checks ensure that the contents of the working registers were not modified during the exponentiation or the recombination computations.

In Sect.4.4, we will review in more detail the previous countermeasures in light of our attack model and show how they can be compromised.

3. NEW COUNTERMEASURE TO FAULT ATTACKS ON CRT-RSA

In this section we present our new FA resistant CRT-RSA algorithm and discuss its correctness.

Algorithm 1. Blinded CRT-RSA Algorithm

```
Input   Private parameters: $p, q, d_p, d_q, i_q = q^{-1} \bmod p$,
        public parameters: $n, e$,
        message (padded and hashed): $m$.
Output  Signature $s$.

 1. Select random values $\rho_p, \rho_q, r_1, r_2, t$ and $\alpha$ at the beginning or right when needed.
 2. $d'_p \leftarrow d_p + \rho_p(p-1)$ // mask the private exponents
 3. $d'_q \leftarrow d_q + \rho_q(q-1)$

 4. $p^* \leftarrow r_1\, p$      // blind the moduli
 5. $q^* \leftarrow r_2\, q$

 6. $b_{p^*} \leftarrow t^{\,\alpha e} \bmod p^*$   // message blinding multipliers
 7. $b_{q^*} \leftarrow t^{\,\alpha e} \bmod q^*$

 8. $m_{p^*} \leftarrow m \bmod p^*$   // message components
 9. $m_{q^*} \leftarrow m \bmod q^*$

10. $\tilde{s}_{p^*} \leftarrow ((m_{p^*}\, b_{p^*})^{(d'_p - 1)}\, m_{p^*}) \bmod p^*$
                    // $\equiv m_p^{d_p}\, t^{\,\alpha(1-e)} \pmod p$
11. $\tilde{s}_{q^*} \leftarrow ((m_{q^*}\, b_{q^*})^{(d'_q - 1)}\, m_{q^*}) \bmod q^*$
                    // $\equiv m_q^{d_q}\, t^{\,\alpha(1-e)} \pmod q$

12. $\tilde{s} \leftarrow (\tilde{s}_{q^*} + q\ (\ i_q\ (\tilde{s}_{p^*} - \tilde{s}_{q^*}) \bmod p^*)) \bmod n$
                    // $\equiv m^d\, t^{\,\alpha(1-e)} \pmod n$

13. $b \leftarrow (b_{q^*} + q\ (\ i_q\ (b_{p^*} - b_{q^*}) \bmod p^*)) \bmod n$
                    // $\equiv t^{\,\alpha e} \pmod n$

14. $s \leftarrow \tilde{s}\, t^{\,\mathrm{trunc}((m\, b\, +\, \alpha(e-1)\, -\, (\tilde{s}\, b)^e) \bmod n)} \bmod n$
                    // $\equiv \tilde{s}\, t^{\alpha(e-1)} \equiv m^d \pmod n$

15. Return($s$)
```

The values ρ_p, ρ_q, r_1, r_2, t and α are random values generated by the algorithm. Suggestions for the sizes of these values are given in Sect. 5. trunc is the truncation function as discussed below.

In the following, we discuss the correctness and purpose of each component of our countermeasure. The resistance of the algorithm to fault attacks is assessed in the next section.

- The RSA moduli p and q are blinded by multiplying them by r_1 and r_2, respectively. This is similar to the blinding in [5, 6, 11, 18, 20, 21]); however, a novelty is that we do not require that r_1 and r_2 be co-prime, and we do not need to compute their Euler totient values. This is based on the following two facts:

  – A value v reduced modulo p^* is congruent to $v_p = v \bmod p$:
  Let $v_p = v \bmod p = v - k_1 p\,; v_p < p$ and $v_{p^*} = v \bmod p^* = v - k_2 p^*\,; v_{p^*} < p^*$ for some integers k_1 and k_2. Reducing v_{p^*} modulo p, we have $v'_p = v_{p^*} \bmod p = v_{p^*} - k_3 p = v - k_2 p^* - k_3 p = v - k_4 p; v'_p < p$ for some integers k_3 and k_4. From the division algorithm, we have $k_1 = k_4$ and $v_p = v'_p$. Therefore,
  $$v_{p^*} \equiv v_p \pmod{p} \ . \qquad (5)$$

  – In a *residue number system*, a value $v \bmod n$, where $n = pq$, can also be expressed by its residues $(v \bmod p,\ v \bmod q)$, when p and q are co-prime.

  For correctness of the RSA-CRT signature, we require a value in residue form (s_p, s_q), where: $s_p = m^{d_p} \bmod p$, and $s_q = m^{d_q} \bmod q$. We calculate instead the signature components modulo the randomized moduli—assuming the message blinding and the exponent randomization, which will be discussed subsequently, were removed: $s_{p^*} = m^{d_p} \bmod p^*$, and $s_{q^*} = m^{d_q} \bmod q^*$. From (5), $s_{p^*} \equiv s_p \pmod{p}$, and similarly $s_{q^*} \equiv s_q \pmod{q}$. We form our combined signature in a Garner-like form:
  $$s = \left\{ s_{q^*} + q \left[(q^{-1} \bmod p)(s_{p^*} - s_{q^*}) \bmod p^* \right] \right\} \bmod n \ , \qquad (6)$$
  where, notably, we do not need to compute q^{*-1} as in [11, 20, 21].

  Using (5), first modulo q, we have $s \equiv s_{q^*} \equiv s_q \pmod{q}$. Similarly, modulo p we have:
  $$\begin{aligned} s &\equiv (s_{q^*} \bmod p) \\ &\quad + (q \bmod p)(q^{-1} \bmod p)\left[s_p - (s_{q^*} \bmod p)\right] \pmod{p} \ , \\ &\equiv (s_{q^*} \bmod p) + \left[s_p - (s_{q^*} \bmod p)\right] \pmod{p} \ , \\ &\equiv s_p \pmod{p} \ . \end{aligned}$$
  Since a final reduction mod n is performed, and because s is correct modulo both p and q, then computing (6) is equivalent to computing $m^d \bmod n$.

  Note that r_1 and r_2 are not used as moduli for auxiliary checks (as they are in [2, 5, 6, 11, 18, 20, 21]). Having no restrictions on the values of r_1 and r_2 conveniently enables choosing new ones for every signature and requires no precomputation. Blinding the RSA moduli in this manner helps prevent *side-channel* attacks (such as the *differential power analysis* (DPA) attack in [13]) and also timing attacks (as in [26]) that exploit the modular reduction modulo p and q. It also protects against the injection of chosen bases designed to aid an attacker in discovering secret exponents.

- The private exponent d_p is randomized by adding a random multiple of $p-1$ to obtain d'_p as in Step 2 [2]. Similarly, d'_q is obtained from d_q. This randomization is useful in thwarting *safe-error* fault attacks [30]. The random multiplier ρ_p and ρ_q are randomly chosen for every signature generation.

  Note that first-order DPA attacks (see *e.g.*, ZEMD in [23]), where an attacker guesses the exponent bits one at a time and computes intermediate values of the exponentiation algorithm, do not apply to the CRT-RSA algorithm since p and q are not available to the attacker for computing those intermediate values [13].

- Antipa's inversionless message blinding [1] is employed to blind the two main exponentiations to prevent timing attacks [26]: Let $s_p = m^{d_p} \bmod p$ be the value required to be computed. A random value r of adequate size is chosen, $r^e \bmod p$ is computed, then $\tilde{s_p} = (r^e m)^{d-\sigma} m^\sigma \bmod p$ is the blinded signature component; $\tilde{s_p} \equiv r^{1-e\sigma} m^d \pmod{p}$. Then $\tilde{s_p}$ is unblinded by computing $\tilde{s_p}\ r^{e\sigma-1} \bmod p$. The public exponent e is used in this blinding; this is generally acceptable since it was suggested in [7] that the device verifies the signature before releasing it by checking whether $s^e \bmod n \overset{?}{=} m$. On the other hand, the full private exponent $d = e^{-1} \bmod (p-1)(q-1)$ is not required as was the case in other countermeasures [5, 6, 21], where it was used in a precomputation phase. It is the use of e that allows our combination of signature verification and unblinding on the signature modulo n; this work could be interpreted as examining the advantages of using e. For example, in the personalization procedure of smart cards in a mass-production, if e is a constant small value, as is usually the case, it may be made available to every card without being part of the personalization parameters.

  In Algorithm 1, where we have set $\sigma = 1$ and $r = t^\alpha$, this message blinding serves the purpose of keeping the signature components, as well as the recombined signature, blinded until it is validated according to the concept proposed by Kim-Quisquater [20, 21] where they employed different blinding methods to prevent attacks that attempt to skip the validation steps of the algorithm [20]. However, the Kim-Quisquater validation values, though computed after the CRT recombination, in essence validate each signature component separately since they are computed modulo the auxiliary moduli that are used to blind p and q; whereas our new validation technique is applied directly to the recombined signature using a recombined validation value as we discuss below. We show in Sect.4.4 how that separate signature validation is vulnerable under our attack model.

  To unblind the recombined signature $\tilde{s}$, we need to compute $s = \tilde{s}\, t^{\alpha(e-1)}$. In order to make this unblinding conditional on the validity of s, the exponent of t is expressed as $f(m, e, \tilde{s}, t, \alpha)$ such that this function evaluates to $\alpha(e-1)$ only if $s \equiv m^d \pmod{n}$ with overwhelming probability. In Step 14, we have
  $$f(m, e, \tilde{s}, t, \alpha) = \mathrm{trunc}((mb + \alpha(e-1) - (\tilde{s}b)^e) \bmod n) \ ,$$
  where, from Step 13, $b \equiv t^{\alpha e} \pmod{n}$ and trunc() is

a function that truncates its input to a bit length no less than that of $\alpha(e-1)$. If no fault occurs, then the following holds

$$\begin{aligned}(\tilde{s}\,b)^e &\equiv (m^d\,t^{\,\alpha(1-e)}\,t^{\,\alpha e})^e && \pmod{n}\ ,\\ &\equiv (m^d\,t^{\,\alpha})^e && \pmod{n}\ ,\\ &\equiv m\,b && \pmod{n}\ .\end{aligned}$$

We recommend additional measures to our algorithm. For example, we can avoid processing the secret values p, q, d_p, d_q, i_q explicitly. Instead, they may be input to the algorithm as randomly split components that are not combined until each component is processed separately in a computation. For example, p may be represented as $p_1 - p_2$ and Step 4 carried out as

$$\begin{aligned}p^* &\leftarrow r_1 p_1\\ p^* &\leftarrow p^* - r_1 p_2\end{aligned}$$

Since all secret components have a random representation in every run, a powerful attacker cannot employ bit-set-reset faults (cf. Sect.4.1) to mount safe-error attacks.

Further countermeasures that seal side-channel information can be integrated. Exponentiation protected against side-channel leakage and fault injection should be employed. Randomization of the computation sequence and/or location is also a valuable additional measure.

4. SECURITY EVALUATION

In this section we provide an overview of previous attack models and present ours. In light of our attack model, we assess the resistance of previous countermeasures then that of Algorithm 1 to various fault attacks.

4.1 Blömer's Attack Models

In [6], Blömer *et al.* gave a collection of attack models, identifying different parameters that define a model:

- control on the fault location,
- control on the fault timing,
- control on the number of bits affected,
- the fault type:
  - stuck at fault,
  - bit flip fault,
  - bit set and reset fault,
  - random fault.

According to the authors, the strongest attack model is the bit-set-reset (bsr) type where the attacker has complete control over the fault location—targeting an exact bit in an exact value—and the timing—injecting the fault at a precise time while running the device on *his* clock. They mention that this is only plausible if the device, *e.g.*, a smartcard, is not equipped with hardware countermeasures such as randomized clocks, memory encryption/decryption schemes, and randomized address scrambling. Aumüller *et al.* [2] also mentioned that current smartcard ICs are equipped with sensors and filters to detect variations in the supply voltage that are outside the tolerable range to prevent voltage spike attacks, which they could switch off in their lab experiment in order to test the validity of their software countermeasure.

In the countermeasure presented by Blömer *et al.*, the validation of the signature was performed via values reduced modulo small primes $t_i, i = 1, 2$. In their security analysis, the random errors yielded higher attack success probability since the faulty values were in the range $[0, t_i)$, whereas the *stronger* attacks, such as the bit or byte errors, yielded lower success probabilities since the faulty values were more restricted. However, in his cryptanalysis using a byte error example, Wagner [29] made use of the predictability of the error pattern since the errors can be enumerated.

Therefore, for every attack model, one should investigate the weakest point of the countermeasure and assess its resistance to a fault at that point.

4.2 Wagner's Framework

The framework introduced by Wagner in [29] is useful in assessing the security of an algorithm. He suggests modeling the state of the device s_i as the contents of the registers and memory, the set of possible states is S. Each step of the algorithm is represented as a relation $\rightsquigarrow$ on $S \times S$. The algorithm is then viewed as a sequence of steps:

$$\langle x, k\rangle = s_0 \rightsquigarrow s_1 \rightsquigarrow \cdots \rightsquigarrow s_n = \langle y\rangle\ ,$$

where x represents the collection of inputs, k that of secrets and y that of outputs. The attack model defines a family of possible faults $\mathcal{F}$ as a relation $\twoheadrightarrow_i$ on $S \times S$. A fault attack is specified by the tuple $(x, \twoheadrightarrow_1, \ldots, \twoheadrightarrow_n)$. An algorithm is secure, if for all the tuples that could be chosen, an attacker cannot learn y or a part thereof. The faulty computation is represented as

$$\langle x, k\rangle = s_0 \rightsquigarrow s_1 \twoheadrightarrow_1 \underline{s_1} \rightsquigarrow s_2 \twoheadrightarrow_2 \underline{s_2} \cdots \rightsquigarrow s_n \twoheadrightarrow_n \underline{s_n} = \langle y\rangle\ .$$

4.3 Our Attack Model

In our model, we assume that the attacker has full control over the timing and the location of the fault, *i.e.*, can target a specific bit or more in a specific variable at a specific step of the algorithm or can opt to inject a random fault to completely change a value. $\mathcal{F}$ comprises the following fault types:

- multiple bit-set-reset (bsr) faults in one or multiple values, in one or multiple steps. We assume that when multiple faults are injected in the same execution step and in the same value, they can alter only a small portion of that value; for example a byte or a word thereof,
- skipping a single or multiple instructions by possibly injecting a fault in the *program counter* (PC) register,
- changing the outcome of a conditional check by injecting a fault in the status register.

This model depicts a powerful attacker injecting faults in a device where hardware countermeasures are not perfect. However, we must assume that the CRT-RSA parameters p, q, d_p, d_q, i_q are protected and are error-free (see also Sect. 4.5).

4.4 Previous Countermeasures Viewed Under Our Model

The most recently proposed countermeasures were Kim-Quisquater's [20, 21] and Boscher-Handschuh-Trichina's (BHT) [9].

The BHT countermeasure modified the initialization of Boscher, Naciri and Prouff's (BNP) [10] exponentiation algorithm in order to render it resistant to DPA attacks. In fact, the BNP exponentiation was based on the right-to-left binary algorithm which originally employed two working registers, performing a squaring operation in every iteration and a multiplication only if the exponent bit is 1. An exponentiation algorithm with uniform iterations is required in order to resist simple side-channel attacks [22]. Hence, a third working register was added in the BNP exponentiation and the multiplication was performed in every iteration. The time performance was then the same as the Montgomery ladder [19] or the square-and-multiply always [12] algorithms but as the authors mention, was 33% faster than the slower Fumaroli-Vigilant algorithm [14] (which employs an additional squaring).

The relation maintained between the three working registers is used at the end to verify that their contents have not been modified. The authors mention that "If the loop is attacked such that it ends before all the bits were processed or if the exponent k is corrupted, this will not be detected, so additional checksums over the inputs should be computed." In the same way, even if the exponent is verified to be unchanged at the beginning of the exponentiation, a bit-set-reset fault that toggles an exponent bit during the course of the exponentiation would go undetected just as would a modified input exponent. Merely the output relation between the registers is checked, but this relation holds for any exponent value if the register contents are unaltered.

The three values of the exponentiation modulo p are CRT-recombined with their counterparts modulo q, then the same check is performed on the recombined values. The countermeasure is therefore vulnerable to exponent bit-set-reset faults *i.e.*, a faulty signature component would be undetected, since the relation between the three recombined values would hold without any dependence on the exponents used to compute its components.

We now discuss Kim-Quisquater's modification of Ciet-Joye's scheme [11, 20], where the authors introduced the importance of validating a blinded signature. In Step 1, the signature and validation component are computed as:

$$s_p^* = (\mathrm{a} + m^{d_p}) \bmod p^*, \quad s_2 = (\mathrm{a} + m^{d_q \bmod \phi(r_2)}) \bmod r_2 \ ,$$
$$s_q^* = (\mathrm{a} + m^{d_q}) \bmod q^*, \quad s_1 = (\mathrm{a} + m^{d_p \bmod \phi(r_1)}) \bmod r_1 \ ,$$

where r_1 and r_2 are relatively small coprime integers, $p^* = r_1 p$, $q^* = r_2 q$ and a is a random integer in $\mathbb{Z}^*_{r_1 r_2 n}$. The recombined signature s^* is validated by computing:

$$c_1 = (s^* - s_1 + 1) \bmod r_1 \ , \quad c_2 = (s^* - s_2 + 1) \bmod r_2 \ .$$

In the case where no fault occurs, we have $c_1 = c_2 = 1$ and the signature is correctly unblinded. Ciet and Joye had stated "The order of the computation is very important: s_p^*, s_2, s_q^* and s_1. If the computation is carried out as s_p^*, s_1, s_q^* and s_2 then a long-lived fault on m before the computation of s_q^* (and after s_1) would go undetected (i.e., $c_1 = c_2 = 1$)"; using the unblinded signature, a gcd attack would yield q. We argue that even with the recommended ordering of operations, an attack can be mounted: If the attacker injects a precise fault in m as it is read for the computation of the signature component s_p^*, and again injects the same fault before the computation of its validation component s_1, then c_1 would evaluate to 1 and the gcd attack is applicable. The same argument can be carried to Kim-Quisquater's scheme in [21] and also to Blömer-Otto's improved BOS [6] scheme in [5]. In general, this attack is expected to apply to schemes that rely on validating the signature components modulo p and q separately, where the validation components are a function of m. Therefore, our new validation technique is performed instead on the recombined signature.

On a side note, we would like to draw the attention to a potential safe-error attack on Kim-Quisquater's exponentiation algorithm in [21], where the authors were improving Fumaroli-Vigilant's exponentiation algorithm [14] due to its vulnerability to fault attacks when used in the CRT-RSA computation. Yet in the improved algorithm [21], the accumulator is multiplied by a value a_{d_i} where d_i is the current exponent bit. To determine the jth bit, an attacker may toggle a precise bit in a_0, for example, at the beginning of the jth iteration and toggle it again before the following iteration and observe whether this toggling had affected the results; if there is no effect then the attacker concludes that $d_j = 1$ since a_0 was not used in this iteration. (Note that bit toggling is possible in our model even if we assume that the attacker can only set/reset a bit, it may just require repeating the attack to discover the modified bit's original value before knowing whether to set it or reset it.) This attack is feasible since the values a_0 and a_1 are constant and do not evolve with the computation as in the Montgomery ladder method [16] cited by the authors.

Another possible attack scenario for a powerful attacker results from expecting constant values as the outcome of a validation step, such as the values $c_1 = c_2 = 1$ as mentioned above or some values being 0, as in [28], where the conditional checks can be converted into infective values, as the author pointed out. We assume that, if small locations are used to store these values, the attacker may be able to set these locations in RAM to their required values. This certainly requires a more controlled attack than setting the status bits in the control register to toggle the outcome of conditional checks.

4.5 Security Assessment of Algorithm 1

As mentioned previously, the strength of Algorithm 1 is that the signature validation is not performed modulo p and q, or separately modulo their blinding values r_1 and r_2 as in [11, 20, 21], but instead modulo n, and after the signature has been CRT-recombined but before the signature has been unblinded. Moreover, the validation computation is infective in nature and also does not yield a constant value.

We should first establish that the CRT-RSA parameters p, q, d_p, d_q, i_q are error-free; for example, using CRC techniques as was proposed by Ciet-Joye [11]. Another possible protective measure is to provide the parameter n, since it would then be computationally infeasible for an attacker to modify p, for example, and modify n accordingly without knowing q. Otherwise, in this case, if an attacker can change p into $\underline{p}$ and n is not provided, but is instead computed internally as $\underline{n} = \underline{p}q$, then a gcd attack is applicable using the original n as in (4).

The question now is how can an attacker inject a fault in a computation modulo one of the moduli only and successfully bypass the final validation so that the gcd attack can be mounted. In order to achieve that goal, the attacker may choose to inject a fault in variables and/or computations modulo p, and not those modulo q, or vice-versa, then inject one or more subsequent faults in order to counteract and/or bypass the blinding of these values. He may target d_p, m_{p^*}, $\tilde{s}_{p^*}$ or their blinding values/computations.

We first focus on injected faults that alter a value in a register or memory location, then we discuss faults that alter the value of the program counter. We are not concerned herein with faults that alter the status register values since Algorithm 1 does not rely on conditional checks.

Following Wagner's framework notation as recalled in Sect. 4.2 and emphasizing only the affected steps of the computation, we examine the following scenario: An attacker injects a fault in d'_p such that $\underline{d'_p} \not\equiv d_p \pmod{p-1}$. Note that if an attacker knows how to modify d'_p into $\underline{d'_p} \equiv d_p \pmod{p-1}$, then the attacker also knows p and has no need for a faulty signature. On the other hand, injecting a random fault that completely modifies d'_p will succeed with probability around $1/p$. Moreover, if the attacker succeeds in making $\underline{d'_p} \equiv d_p \pmod{p-1}$, then the scheme is not affected, and there will be no fault in $\tilde{s}_{p^*}$ that can enable a gcd attack.

$$\begin{aligned}
d'_p \twoheadrightarrow_1 \quad \underline{d'_p} &\not\equiv d_p && \pmod{p-1} \ , \\
\rightsquigarrow \quad \underline{\tilde{s}_{p^*}} &\not\equiv \tilde{s}_{p^*} && \pmod{p} \ , \\
\rightsquigarrow \quad \underline{\tilde{s}} &\equiv m^{\underline{d}}\, t^{\,\alpha(1-e)} && \pmod{n} \ , \\
\rightsquigarrow \ (\underline{\tilde{s}}b)^e &\equiv \underline{m}b && \pmod{n} \ , \\
\rightsquigarrow \quad s &\equiv \underline{\tilde{s}}t^{\,\mathrm{trunc}(mb-\underline{m}b)+\alpha(e-1))\bmod n)} && \pmod{n} \ , \\
&\equiv m^{\underline{d}}\, t^{\beta} && \pmod{n} \ , \\
&\equiv m^{d_q}\, t^{\beta} && \pmod{q} \ , \\
&\not\equiv s_{q^*} && \pmod{q} \ ,
\end{aligned}$$

where $b = t^{\,\alpha e}$ and $\beta = \underline{m}b - mb \mod n$. Can the attacker, by injecting other precise faults, modify any of the values of the algorithm to reduce β to 0? This would require modifying m in Step 14 to a precise value that the attacker cannot compute without knowing b or $\underline{m}$. More generally, is it possible to inject a combination of precise faults that would result in $s \not\equiv s_p$ while still $s \equiv s_q$? As another illustration, an attacker can apply the same attack as the one we demonstrated on Kim-Quisquater's [20, 21] in 4.4: The attacker can inject a fault in m_{p^*} in 8, which would result in the following sequence

$$\begin{aligned}
m_{p^*} \twoheadrightarrow_1 \quad \underline{m_{p^*}} &\not\equiv m_p && \pmod{p} \ , \\
\rightsquigarrow \quad \underline{\tilde{s}_{p^*}} &\not\equiv \tilde{s}_{p^*} && \pmod{p} \ , \\
\rightsquigarrow \quad \underline{\tilde{s}} &\equiv m^{\underline{d}}\, t^{\,\alpha(1-e)} && \pmod{n} \ , \\
\rightsquigarrow \quad (\underline{\tilde{s}}b)^e &\equiv \underline{m}b && \pmod{n} \ , \\
\rightsquigarrow \quad s &\equiv \underline{m}^d\, t^{\beta} && \pmod{n} \ , \\
&\equiv m^{d_q}\, t^{\beta} && \pmod{q} \ , \\
&\not\equiv s_{q^*} && \pmod{q} \ ,
\end{aligned}$$

which leaves the attacker in the same position, needing to modify m in Step 14 to an unknown value. Attacks on $\tilde{s}_{p^*}$ directly result in similar conclusions; the attacker has the power to alter any variables but has no knowledge of their expected values.

Our validation is performed on the recombined signature itself, not on each modular component, as in the previous countermeasures [2, 5, 6, 11, 18, 20, 21]. This yields a new strength: when our infective validation fails, it infects both signature components.

If the attacker opts to inject a random fault, then the smallest candidate value to change is the exponent in Step 14 which is the outcome of the truncation function. The bit length of that outcome, then, is a security parameter that depends on the bit length of e and α. Specifically, α is chosen based on a compromise between performance and security (see bit-length suggestions in Sect. 5).

Can the attacker modify the sequence of the algorithm in order to avoid countermeasure instructions? For example, can the attacker skip the multiplications instructions $m_{p^*}\ b_{p^*} \mod p^*$ and $m_{q^*}\ b_{q^*} \mod q^*$ that blind the message and then skip the unblinding multiplication in Step 14? We assume this is possible. However, with careful implementation, it can be ensured that the result would be an unpredictable value:

To illustrate this design concept, let registers A and B hold the values m_{p^*} and b_{p^*}, respectively. Then, register A should not be the target of the multiplication, otherwise, if it is skipped, then it would hold the same value without applying the blinding. If we let the target register be C, we should make certain that C contains no values of sensitive nature that would aid the attacker. Along the same lines, we should ensure that the final register/location from which the signature is copied to the output buffer does not contain any sensitive values produced during the execution of the entire algorithm. We must also assume here that register addressing operations are sufficiently protected to render the attacker unable to precisely control which register locations are being manipulated.

5. PERFORMANCE EVALUATION

Regarding the time requirement of Algorithm 1, as was mentioned in Sect. 3, r_1 and r_2 are not chosen as prime or even coprime integers [5, 6, 11, 18, 20, 21]); hence, there is no requirement to compute their totient values as in the cited countermeasures that include a random multiplier of p and q. Moreover, there is no inversion required modulo the RSA moduli or their multiplier(s) as was the case in several countermeasures [5, 6, 11, 21, 28]. Therefore, there is no need for a precomputation phase. Nevertheless, Steps 2 to 7 can be performed before the next message to be signed is available.

The following are the overhead computations compared to an unprotected CRT-RSA scheme, where $|a|$ denotes the bit length of a value a (Suggested bit lengths for random values are typical for 1024-bit RSA; they are large enough in order to thwart multiple attacker attempts without much affecting the time and memory requirements):

1. A general overhead of $\mathcal{O}(|r_i|^2)$ where $|r_i| = \max(|r_1|, |r_2|)$ in all modular multiplications modulo the prime mod-

uli, as a result of blinding; r_1 and r_2 may be 32 bits long (*e.g.*, [17, 27]).

2. An overhead of $\mathcal{O}(|\rho||r_i|^2)$ where $|\rho| = \max(|\rho_p|, |\rho_q|)$ in the two main exponentiations resulting from the exponent blinding; ρ_p and ρ_q may be 32 bits long.
3. Two exponentiations modulo p^* and q^*, respectively, where the exponent is αe and t is a random value of the same bit length as p and q. For example, if $e = 2^{16} + 1$ then α may be 15 bits long. Note that if we eliminate the assumption that an attacker can directly set the value of the validation exponent in Step 14, then we can set $\alpha = 1$. The validation exponent would then always be expected to have the value $e - 1$.
4. Four modular multiplications in Steps 10 and 11(which include the overhead mentioned in item 1 above).
5. A CRT recombination in Step 13.
6. A multiplication and an exponentiation modulo n in Step 14 where the exponent is of bit length $|\alpha e|$.

The blinding overhead, including the cost of randomness generation, is typical for all protected CRT-RSA implementations. In fact, the message blinding technique we adapted [1] is more advantageous since it requires no inversion as in other countermeasures [14, 21]. The main new overhead we introduce is in the implicit verification in Step 14. For example, for a 1024-bit n, $e = 2^{16} + 1$ and a 15-bit α, this overhead is roughly 12.5% of the unprotected CRT exponentiations. For a 2048-bit n, the overhead is reduced to 6.25%, since e is proportionally smaller.

As for the space requirement, the extra values that would need to be stored are b_q and b_p which can be later replaced by b. Other values to be stored are t and α which are typically used in countermeasures that blind the message [20, 21]. Also the extension of the prime moduli should be taken into account in the extra storage needed, if employed.

6. CONCLUSION

In this paper, we have presented a CRT-RSA countermeasure designed to resist a powerful attacker who can inject bit-set-reset faults in the same execution step or across different steps, control the algorithm clocking and target specific small locations. We have also assumed that the attacker can skip one or more intermediate instructions and can alter the result of conditional checks.

We have presented and analysed the performance and security of a new CRT-RSA countermeasure:

- The scheme allows easy randomization for each signature, since there are no requirements on the random values used to blind the moduli, the exponents or the message.
- The blinding computation does not require precomputation.
- The CRT recombination does not require the computation of q^{*-1}, *i.e.*, the inverse of one of the blinded moduli.
- Unlike previous countermeasures, the validation of the blinded signature is implicitly performed modulo n after the CRT recombination rather than prior to it. In case the validation fails, the entire blinded signature is infected and the unblinding yields a random value modulo both p and q. Hence, it is infeasible for a powerful attacker to precisely inject one or more faults aiming at modifying one signature component without affecting the other one and mounting a gcd attack.

References

[1] A. Antipa. Method and apparatus for exponentiation in an RSA cryptosystem. United States Patent #7,177,423, February 13, 2007.

[2] C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert. Fault attacks on RSA with CRT: Concrete results and practical countermeasures. In *Cryptographic Hardware and Embedded Systems – CHES '02*, volume 2523 of *LNCS*, pages 260–275. Springer-Verlag, 2002.

[3] H. Bar-el, H. Choukri, D. N. M. Tunstall, and C. Whelan. The sorcerer's apprentice guide to fault attacks. In *Workshop on Fault Diagnosis and Tolerance in Cryptography - FDTC '04 in association with the International Conference on Dependable Systems and Networks - DSN '04*, pages 330–342, 2004.

[4] H. Bar-el, H. Choukri, D. N. M. Tunstall, and C. Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2):370–382, Feb. 2006. An earlier version of this paper appeared in [3]. Also available at `http://eprint.iacr.org/2004/100.pdf`.

[5] J. Blömer and M. Otto. Wagner's attack on a secure CRT-RSA algorithm reconsidered. In *Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC '06*, volume 4236 of *LNCS*, pages 13–23. Springer-Verlag, 2006.

[6] J. Blömer, M. Otto, and J.-P. Seifert. A new CRT-RSA algorithm secure against Bellcore attacks. In *ACM Conference on Computer and Communications Security – CCS '03*, pages 311–320. ACM Press, 2003.

[7] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of eliminating errors in cryptographic computations. In *Advances in Cryptology – EUROCRYPT '97*, volume 1233, pages 37–51. Springer-Verlag, 1997. This paper appeared after a Bellcore press release in 1996, for example see `http://findarticles.com/p/articles/mi_m0EIN/is_1996_Sept_26/ai_18720539/pg_1`.

[8] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of eliminating errors in cryptographic computations. *Journal of Cryptology*, 14:101–119, 2001. This is an expanded version of an earlier paper that appeared in [7].

[9] A. Boscher, H. Handschuh, and E. Trichina. Blinded fault resistant exponentiation revisited. In *Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC '09*, pages 3–9. IEEE Computer Society, 2009.

[10] A. Boscher, R. Naciri, and E. Prouff. CRT RSA algorithm protected against fault attacks. In *Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems – WISTP '07*, volume 4462 of *LNCS*, pages 229–243. Springer-Verlag, 2007.

[11] M. Ciet and M. Joye. Practical fault countermeasures for Chinese remaindering based RSA. In *Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC '05*, pages 124–131, 2005.

[12] J.-S. Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In *Cryptographic Hardware and Embedded Systems – CHES '99*, volume 1717 of *LNCS*, pages 292–302. Springer-Verlag, 1999.

[13] B. den Boer, K. Lemke, and G. Wicke. A DPA attack against the modular reduction within a CRT implementation of RSA. In *Cryptographic Hardware and Embedded Systems – CHES '02*, volume 2523 of *LNCS*, pages 228–243. Springer-Verlag, 2002.

[14] G. Fumaroli and D. Vigilant. Blinded fault resistant exponentiation. In *Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC '06*, volume 4236 of *LNCS*, pages 62–70. Springer-Verlag, 2006.

[15] H. L. Garner. The residue number system. *IRE Transactions on Electronic Computers*, 8(6):140–147, 1959.

[16] C. Giraud. Fault resistant RSA implementation. In *Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC '05*, pages 142–151, 2005.

[17] C. Giraud. An RSA implementation resistant to fault attacks and to simple power analysis. *IEEE Transactions on Computers*, 55(9):1116–1120, 2006. An earlier version appeared in [16].

[18] M. Joye, P. Paillier, and S.-M. Yen. Secure evaluation of modular functions. In *Cryptology and Network Security – CNS '01*, pages 227–229, 2001.

[19] M. Joye and S.-M. Yen. The Montgomery powering ladder. In *Cryptographic Hardware and Embedded Systems – CHES '02*, volume 2523 of *LNCS*, pages 291–302. Springer-Verlag, 2002.

[20] C. H. Kim and J.-J. Quisquater. Fault attacks for CRT based RSA: New attacks, new results, and new countermeasures. In *Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems – WISTP '07*, volume 4462 of *LNCS*, pages 215–228. Springer-Verlag, 2007.

[21] C. H. Kim and J.-J. Quisquater. How can we overcome both side channel analysis and fault attacks on RSA-CRT? In *Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC '07*, pages 21–29. IEEE Computer Society Press, 2007.

[22] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Advances in Cryptology – CRYPTO '99*, volume 1666 of *LNCS*. Springer-Verlag, 1999.

[23] T. S. Messerges, E. A. Dabbish, and R. H. Sloan. Power analysis attacks of modular exponentiation in smart cards. In *Cryptographic Hardware and Embedded Systems – CHES '99*, volume 1717 of *LNCS*, pages 144–157. Springer-Verlag, Aug. 1999.

[24] J.-J. Quisquater and C. Couvreur. Fast decipherment algorithm for RSA public-key cryptosystem. *IEE Electronics Letters*, 18(21):905–907, 1982.

[25] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978. An earlier version appeared as Technical Report MIT/LCS/TM-82 (1977).

[26] W. Schindler. A timing attack against RSA with the Chinese Remainder Theorem. In *Cryptographic Hardware and Embedded Systems – CHES '00*, volume 1965 of *LNCS*, pages 109–124. Springer-Verlag, 2000.

[27] A. Shamir. Method and apparatus for protecting public key schemes from timing and fault attacks. United States Patent #5,991,415, November 23, 1999. Presented earlier at the rump session of EUROCRYPT '97.

[28] D. Vigilant. RSA with CRT: A new cost-effective solution to thwart fault attacks. In *Cryptographic Hardware and Embedded Systems – CHES '08*, volume 5154 of *LNCS*, pages 130–145. Springer-Verlag, 2008.

[29] D. Wagner. Cryptanalysis of a provably secure CRT-RSA algorithm. In *ACM Conference on Computer and Communications Security – CCS '04*, pages 92–97. ACM Press, 2004.

[30] S.-M. Yen and M. Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Transactions on Computers*, 49(9):967–970, 2000.

[31] S.-M. Yen, S. Kim, S. Lim, and S.-J. Moon. RSA speedup with residue number system immune against hardware fault cryptanalysis. In *Information Security and Cryptology – ICISC '01*, pages 397–413, 2001.

[32] S.-M. Yen, S. Kim, S. Lim, and S.-J. Moon. RSA speedup with Chinese Remainder Theorem immune against hardware fault cryptanalysis. *IEEE Trans. Computers*, 52(4):461–472, 2003. An earlier version of this paper appeared in [31].